

# Ecommerce Starter Template

A Stripe-powered storefront template for `create-profound-next`, with checkout as a headless UI component and a catalog modeled entirely in the Profound CMS admin panel.

PRODUCT

**create-profound-next**

AUTHOR

**Daksh (daksh@tryprofound.com)**

STATUS

**Draft for review**

FEATURE

**Ecommerce + Stripe template**

DATE

**June 13, 2026**

VERSION

**0.1**

## 1. Overview

---

`create-profound-next` today scaffolds a generic Next.js + Profound CMS starter. This PRD proposes a second, opinionated **ecommerce template** that ships a complete, deployable storefront: a Stripe-backed checkout exposed as a **headless UI component**, plus a product catalog (**categories** and **items**) modeled as first-class components inside the Profound CMS admin panel.

The goal is to let a developer run a single command and get a working store where a non-technical merchant can create categories, add products, set prices, and publish — with payments handled by Stripe and the entire visual layer editable in the CMS.

## 2. Problem & Goals

---

### Problem

Building an ecommerce site on a CMS usually means stitching together a payments provider, a product data model, a checkout UI, and a content-editing experience by hand. There is no turnkey path from "scaffold" to "merchant-editable store with real payments."

### Goals

- Scaffold a production-shaped storefront in one command ( `npx create-profound-next --template ecommerce` ).
- Model the catalog (categories + items) as CMS components a merchant edits without touching code.
- Ship Stripe checkout as a **headless** component — logic and state provided, styling owned by the developer.
- Keep the template thin, typed, and consistent with the existing base template's conventions.

### Non-Goals

- A full order-management / fulfillment dashboard (Stripe Dashboard is the source of truth for orders v1).
- Inventory reservations, multi-currency tax engines, or subscriptions (future iterations).
- Replacing Stripe Checkout — we wrap it, we don't rebuild the PCI surface.

### 3. Target Users

Persona	Needs	Touchpoint
Developer	Fast scaffold, typed components, clear Stripe wiring, control over styling.	CLI + codebase
Merchant / Editor	Create categories & products, set prices and images, publish — no code.	Profound CMS admin panel
Shopper	Browse catalog, add to cart, pay securely.	Storefront

## 4. CMS Components (Admin Panel)

Two new CMS components are registered in the catch-all route's `registry` (see `[...slug]/page.tsx`) and exposed in the admin panel for merchant editing.

### 4.1 Category component

Field	Type	Notes
<code>name</code>	text	Display name, e.g. "Apparel"
<code>slug</code>	text	URL segment, unique
<code>description</code>	rich text	Optional intro copy
<code>heroImage</code>	image	Optional banner
<code>items</code>	reference[]	Ordered list of Item components

### 4.2 Item (product) component

Field	Type	Notes
<code>name</code>	text	Product title
<code>slug</code>	text	URL segment, unique
<code>description</code>	rich text	Product detail copy
<code>images</code>	image[]	Gallery
<code>price</code>	number	Minor units (cents)
<code>currency</code>	select	ISO 4217, default <code>usd</code>
<code>stripePriceId</code>	text	Maps to a Stripe Price; source of truth for charging
<code>category</code>	reference	Parent Category
<code>active</code>	boolean	Hide/show without deleting

**Pricing integrity:** the displayed `price` is for presentation; the actual charge is always derived server-side from `stripePriceId`. This prevents client-tampered amounts and keeps Stripe as the pricing source of truth.

## 5. Headless Stripe Checkout Component

Checkout ships as a **headless** component: it owns cart state, line-item validation, and the call to create a Stripe Checkout Session, but renders no opinionated markup. The developer supplies the UI via render props / children, so it adapts to any design.

### 5.1 Shape

```
// headless: logic + state, zero styling
const {
  items,           // cart line items
  addItem, removeItem, updateQty,
  subtotal,        // derived, display-only
  checkout,        // () => redirects to Stripe Checkout
  status,          // 'idle' | 'loading' | 'redirecting' | 'error'
  error,
} = useCart();

<Checkout>
  {( { checkout, status } ) => (
    <button onClick={checkout} disabled={status === 'loading'}>
      Pay
    </button>
  ) }
</Checkout>
```

### 5.2 Server flow

1. Client calls a Server Action / route handler with item ids + quantities.
2. Server resolves each id to its `stripePriceId` from the CMS, ignoring any client-sent price.
3. Server creates a Stripe Checkout Session and returns the redirect URL.
4. Client redirects to Stripe-hosted checkout; Stripe handles card entry (PCI scope stays with Stripe).
5. A `/api/stripe/webhook` handler verifies `checkout.session.completed` and is the trusted fulfillment signal.

**Why headless:** ecommerce designs vary wildly. By providing state + Stripe wiring but no markup, the template stays reusable across brands while keeping the secure path (price resolution, session creation, webhook verification) fixed and correct.

## 6. Architecture & File Additions

Built on the existing base template ( `ParametricRoutePage` , `Refresher` , `createCmsProxy` ). New files added under `src/templates/ecommerce/` :

```
src/templates/ecommerce/
src/
  app/
    [...slug]/page.tsx      # registry += { Category, Item }
    cart/page.tsx          # cart + headless <Checkout>
    api/stripe/
      checkout/route.ts    # create Checkout Session
      webhook/route.ts     # verify + fulfill
  components/
    Category.tsx           # CMS component
    Item.tsx               # CMS component
    ProductGrid.tsx
  lib/
    stripe.ts              # server Stripe client
    useCart.ts             # headless cart hook
  next.config.mjs
  _gitignore
  tsconfig.json
```

The CLI's `generatePackageJson()` adds `stripe` and `@stripe/stripe-js` alongside the existing `cms-renderer` dependency when the ecommerce template is selected.

## 7. Environment Variables

Variable	Scope	Purpose
<code>PROFOUND_API_KEY</code>	server	Existing — CMS access
<code>NEXT_PUBLIC_PROFOUND_WEBSITE_ID</code>	client	Existing — website id
<code>STRIPE_SECRET_KEY</code>	server	Create sessions, verify webhooks
<code>STRIPE_WEBHOOK_SECRET</code>	server	Verify webhook signatures
<code>NEXT_PUBLIC_STRIPE_PUBLISHABLE_KEY</code>	client	Stripe.js redirect
<code>NEXT_PUBLIC_SITE_URL</code>	client	Checkout success/cancel URLs

## 8. Requirements (MoSCoW)

#	Requirement	Priority
R1	CLI scaffolds the ecommerce template via a flag	MUST
R2	Category & Item registered as editable CMS components	MUST
R3	Headless cart hook + Checkout component	MUST
R4	Server-side price resolution from <code>stripePriceId</code>	MUST
R5	Signed Stripe webhook handler for fulfillment	MUST
R6	Category & product listing/detail pages via CMS routing	SHOULD
R7	Cart persistence across reloads (localStorage)	SHOULD
R8	Discount/promo code support	COULD
R9	Inventory count + sold-out state	COULD

## 9. Success Metrics

- **Time-to-first-store:** < 15 minutes from scaffold to a test-mode purchase completing.
- **Zero-code catalog edit:** a merchant creates a category + product and sees it live without a developer.
- **Checkout integrity:** 100% of charges derive their amount server-side from Stripe, never the client.
- **Adoption:** ecommerce template selected in a meaningful share of scaffolds within one quarter of launch.

## 10. Risks & Open Questions

Item	Mitigation / Question
CMS price vs. Stripe price drift	Treat Stripe as source of truth; optional sync script to push CMS price → Stripe Price.
Webhook reliability in local dev	Document <code>stripe listen</code> / Stripe CLI in template README.
Headless API surface scope	Open: how much UI (e.g. a default unstyled cart) should ship as an optional example?
Catalog vs. Stripe Product sync	Open: auto-create Stripe Products from CMS Items, or require manual mapping in v1?

## 11. Milestones

Phase	Scope
M1 — Data model	Category + Item components, registry wiring, listing/detail pages
M2 — Payments	Headless cart, checkout session route, webhook handler
M3 — CLI integration	<code>--template ecommerce</code> flag, deps in <code>generatePackageJson()</code>
M4 — Polish	README, env docs, test-mode walkthrough, example styling

---

create-profound-next — Ecommerce + Stripe Template PRD · v0.1 · Draft for review · June 13, 2026